Design Inspection/Walkthrough Checklist

Number: 580-CK-058-01
Effective Date: April 24, 2006
Expiration Date: April 24, 2011

Approved By: (signature)
Name: Barbara Pfarr
Title: Assoc. Chief, ISD

Responsible Office: 580/Information Systems Division (ISD)
Title: Design Inspection/Walkthrough

Asset Type: Checklist
PAL Number: 2.3.1.3

Design Inspection/Walkthrough Checklist

The Design Inspection/Walkthrough Checklist uses a set of design measures applied to the software design. These measures are characteristics of structural factors that are judged as adequate or not, rather than quantitatively measured and compared against an absolute standard.

Guidance: Also see the Testing Process. Test Case development typically occurs concurrently with design development and the two activities may influence each other.

Considerations to be checked for all designs include:

<table>
<tr><th></th><th></th><th>✓</th><th>Observations and Comments
(Mandatory)</th></tr>
<tr><td>1</td><td><b>Completeness</b> – Specification of design is to the lowest appropriate level.
<table>
<tr><td><i>Guidance: Not all may be applicable for a particular system (e.g., not all systems will need to consider COTs), but each check should be considered.</i></td><td>✓</td></tr>
<tr><td>Review Requirements Traceability Matrix to ensure coverage of all requirements</td><td>□</td></tr>
<tr><td>Ensure coverage of:</td><td></td></tr>
<tr><td>Real-time requirements</td><td>□</td></tr>
<tr><td>Performance issues (memory and timing)</td><td>□</td></tr>
<tr><td>Spare capacity (CPU and memory)</td><td>□</td></tr>
<tr><td>Maintainability</td><td>□</td></tr>
<tr><td>Understandability</td><td>□</td></tr>
<tr><td>Database requirements</td><td>□</td></tr>
<tr><td>Loading and initialization</td><td>□</td></tr>
<tr><td>Error handling and recovery</td><td>□</td></tr>
<tr><td>User interface issues</td><td>□</td></tr>
<tr><td>Software upgrades</td><td>□</td></tr>
<tr><td>Software re-use and modifications</td><td>□</td></tr>
<tr><td>COTS</td><td>□</td></tr>
<tr><td>All inputs and outputs</td><td>□</td></tr>
<tr><td>Clearly and correctly identify interfaces</td><td>□</td></tr>
<tr><td>All functions clearly and accurately described in sufficient detail</td><td>□</td></tr>
<tr><td>All interfaces clearly and (appropriately) precisely defined</td><td>□</td></tr>
<tr><td>Adequate data structures defined</td><td>□</td></tr>
<tr><td>All error codes documented</td><td>□</td></tr>
</table>
</td><td>□</td><td></td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td><b>Suitability</b>– The design itself is good.
<table><tr><td></td><td>✓</td></tr><tr><td>Deviations from the requirements are documented and approved</td><td>□</td></tr><tr><td>Assumptions are documented</td><td>□</td></tr><tr><td>Major design decisions are documented</td><td>□</td></tr><tr><td>The design is expressed in precise unambiguous terms</td><td>□</td></tr><tr><td>Dependencies on other functions, operating system, hardware etc. are documented</td><td>□</td></tr><tr><td>The design follows notational conventions</td><td>□</td></tr></table></td><td>□</td><td></td></tr>
<tr><td>3</td><td><b>Correctness</b> – The design will lead to good software.
<table><tr><td></td><td>✓</td></tr><tr><td>The logic is correct</td><td>□</td></tr><tr><td>Memory and timing budgets are reasonable and achievable</td><td>□</td></tr><tr><td>Error messages are helpful and understandable</td><td>□</td></tr><tr><td>The design is understandable (i.e., easy to read, to follow logic)</td><td>□</td></tr><tr><td>It is maintainable (i.e., no obscure logic);</td><td>□</td></tr><tr><td>It is testable</td><td>□</td></tr><tr><td>It is consistent (i.e., program flow and data format match between sending and receiving components/software units)</td><td>□</td></tr><tr><td>It is cohesive (i.e., proper groupings of related components/functions)</td><td>□</td></tr><tr><td>It is mutually suspicious (i.e., the components/software units check each other for errors in parameters or other exchanged data)</td><td>□</td></tr><tr><td>COTS and GOTS have been verified to fulfill their intended purpose</td><td>□</td></tr></table></td><td>□</td><td></td></tr>
<tr><td>4</td><td><b>Simplicity</b> – Complexity no more than necessary</td><td>□</td><td></td></tr>
</table>

5	**Quality** – A high quality design of a high quality system (nested) Have alternate design approaches been evaluated and the optimum chosen? □ User interface/screens have been verified with end users? □ Are there minimal requirements TBD's? □	□	

	✓
Have alternate design approaches been evaluated and the optimum chosen?	□
User interface/screens have been verified with end users?	□
Are there minimal requirements TBD's?	□

Notes/Action Items for follow-up

#	Action	Assignee	Due Date

References

- Page-Jones, M., The Practical Guide to Structured Systems Design, second edition, Yourdon Press (Prentice-Hall), 1988.
- Ward, P. T. and Mellor, S. J., Structured Development for Real-Time Systems, 3 volumes, Yourdon Press, 1985, 1986.
- Yourdon, E. and Constantine, L., Structured Design, Prentice-Hall, 1979.
- Software Verification and Validation: A Practitioner's Guide by Steven R. Rakitin Artech House © 1997
- Booch, Grady, Object-Oriented Design: With Applications, Benjamin/Cummings, Redwood City, CA, 1991.
- What Makes a Good Object-Oriented Design (http://ootips.org/ood-principles.html)

Change History

Version	Date	Description of Improvements
1.0	4/24/06	Initial approved version by CCB